Mécanisme de régulation dynamique de la charge pour un solveur de Vlasov semi-Lagrangien

Rapport de stage

Olivier Hoenen

DEA d'informatique 2003/2004
Image et Calcul Parallèle Scientifique

Stage encadré par Éric Violard

Table des matières

Remerciements

Je remercie Éric Violard, pour son encadrement sur ce stage, sa disponibilité, et pour m'avoir fait bénéficier de son expérience et de ces conseils. Je tiens aussi à remercier Michel Mehrenberger pour ses précieuses explications sur la méthode numérique. Enfin je remercie les membres de l'équipe ICPS et du projet CALVI pour leur accueil.

Introduction

Ce travail prend place au sein du projet CALVI [1] consacré à l'étude mathématique et numérique et à la visualisation de divers problèmes issus essentiellement de la physique des plasmas et des faisceaux de particules. Ces problèmes sont régis par l'équation de Vlasov.

Un des objectifs essentiel du projet est de développer des outils de simulation et de visualisation robustes et efficaces de ces problèmes physiques.

Les difficultés sont dûes à la présence d'échelles multiples, l'évolution en temps et surtout la grande taille des problèmes à traiter. En effet les problèmes considérés sont posés dans l'espace des phases, i.e l'espace des positions et vitesses, soit un espace à 6 dimensions dans le cas réel ($d = 3$).

Effectuer de telles simulations requiert des méthodes mathématiques et informatiques évoluées dont la mise en oeuvre est non triviale. On s'intéresse plus particulièrement au schéma semi-Lagrangien basé sur les caractéristiques de l'équation de Vlasov. Ce schéma engendre des méthodes qui possèdent des bonnes propriétés en terme de description des systèmes physiques et d'extensibilité des codes parallèles induits. Cependant, pour obtenir des simulations réalistes, il est nécessaire de réduire encore la quantité de calculs. Dans ce contexte, les méthodes dites *adaptatives* sont particulièrement intéressantes. Elles discrétisent l'équation de Vlasov sur un maillage non-uniforme de l'espace des phases et adaptent ce maillage à l'évolution des particules dans le temps.

Le travail présenté porte sur la parallélisation de ces méthodes adaptatives. Elle consiste à distribuer le maillage erntre les processeurs. Chaque processeur réalise alors le traitement d'une partie du maillage. Ainsi les meilleures performances sont obtenues lorsque quand chacune des parties du maillage représente une charge de calcul équivalente, on parle alors d'équilibre de charge. Puisque le maillage évolue au cours du temps, un mécanisme de régulation dynamique est nécessaire pour maintenir l'équilibre de charge. Nos travaux portent sur ce mécanisme et son adaptation à la méthode numérique utilisée.

Ce document est organisé de la manière suivante : La première partie est un tour

[1] CALcul scientifique et VIsualisation, projet INRIA (responsable : Eric Sonnendrücker)

d'horizon des méthodes numériques utilisées pour traiter des problèmes en physique des plasmas et des faisceaux de particules. Dans une seconde partie, une version adaptative de la méthode semi-Lagrangienne de résolution de l'équation de Vlasov et ses fondements mathématiques sont présentés. La parallélisation de cette méthode fait l'objet du troisième chapitre. Le chapitre suivant présente la stratégie d'équilibrage de charge dynamique que nous avons développé. Enfin les résultats numériques des mécanismes implantés sont exposés dans la dernière partie.

Chapitre 1

État de l'art

On va s'intéresser dans un premier temps au modèle des systèmes physiques considérés puis aux méthodes mathématiques développées pour la simulation de ces systèmes physiques. Puis nous présenterons les techniques classiques de parallélisation de telles méthodes, en s'attardant tout particulièrement sur l'équilibrage de charge.

1.1 Modèlisation des systèmes physiques considérés

L'état plasma peut être considéré comme le quatrième état de la matière, obtenu par exemple en portant un gaz à très haute température ($104°K$ ou plus). L'énergie d'agitation thermique des molécules et atomes constituant le gaz est alors suffisante pour déclencher le phénomène d'ionisation lors des collisions entre ces particules. On obtient alors un gaz globalement neutre composé de particules chargées et de particules neutres que l'on appelle plasma.

La physique de base pour modéliser les phénomènes collectifs dans les plasmas est bien connue : on utilise la description cinétique, où le plasma est décrit par une fonction de distribution $f(x, v, t)$. Celle-ci correspond à une moyenne statistique sur un grand nombre de réalisations du système physique considéré. Le produit $fdxdv$ est le nombre moyen de particules de l'espèce considérée dont la position et la vitesse se trouvent respectivement dans un élément de volume dx de l'espace ordinaire et dans un élément de volume dv de l'espace des vitesses, centrés respectivement autour des valeurs x et v.

L'équation cinétique correspondante est l'équation de Vlasov (voir [32] pour plus de détails), donnée ici en variables adimensionnées, c'est-à-dire que les grandeurs physiques (la masse, la charge, etc ...) sont normalisés à 1 :

$$\partial_t f + v \cdot \partial_x f + E \cdot \partial_v f = 0 \tag{1.1}$$

où le champ électrique E est donné par un champ externe E_a et un champ électrique auto-consistant E_s solution de l'équation de Poisson :

$$\partial_x E_s = \rho, \tag{1.2}$$

avec la densité de charge

$$\rho(t,x) = \int_{-\infty}^{\infty} f(t,x,v)dv. \tag{1.3}$$

1.2 Méthodes de résolution

On se référera à [13] pour un résumé de nombreux solveurs pour l'équation de Vlasov. Ces méthodes se regroupent en deux classes principales.

1.2.1 Méthode particulaires

Actuellement, les solveurs de l'équation de Vlasov les plus populaires sont basés sur la méthode PIC (*Particle In Cell*). Elle consiste à décrire l'évolution en temps de l'équation à travers un nombre fini de particules. Elle utilise un maillage de l'espace des positions pour calculer la densité de charge, en faisant une interpolation linéaire de la proportion de particules sur chaque maille [5].

1.2.2 Méthodes non particulaires

Pour certains problèmes en physique des plasmas et des faisceaux, les méthodes particulaires sont trop bruitées (le "bruit numérique" peut provoquer une erreur de l'ordre de $1/\sqrt{N}$, où N est le nombre de particules). On a alors intérêt à résoudre l'équation sur un maillage uniforme de l'espace des phases. Ce maillage peut être structuré [14] ou non-structuré [4], c'est-à-dire que les mailles peuvent être définies ou quelconques. Ces méthodes, envisageables seulement récemment grâce à l'essort de la puissance de calcul des ordinateurs, sont efficaces pour un espace des phases à deux dimensions ($d = 1$) [32].

Parmis ce type de résolution, on trouve des méthodes basées sur le schéma semi-Lagrangien [32, 4]. Elles utilisent la propriété de conservation de la distribution des particules le long des caractéristiques de l'équation de Vlasov. Les valeurs de la fonction de distribution sont donc calculées à partir des valeurs du maillage à l'étape de temps précédente en "remontant" le long des caractéristiques. Des travaux montrent l'intérêt de ces méthodes quant à la qualité de la description du système physique.

Cependant quand la dimension augmente, le nombre de points de discrétisation (ou nombre de points du maillage uniforme) devient beaucoup trop important pour que la

simulation puisse être effectuée par un seul processeur. Deux approches ont été exploitées pour palier à ces problèmes : les méthodes adaptatives et les méthodes par maillage mobile.

Les méthodes adaptatives

Dans cette approche on conserve uniquement les points du maillage qui sont "nécessaires". En effet, en présence d'échelles multiples, certaines zones du maillage sont à fort gradient[1] alors que d'autres en ont un très faible. On peut donc se permettre d'éliminer des points de discrétisation dans les zones où le gradient est faible, et en rajouter dans les zones où il est élevé (*raffinement* du maillage). De telles méthodes, que l'on appelle *méthodes adaptatives*, ont récemment été développées.

Une analyse multi-résolution [1, 9], ou MRA, permet de sélectionner les mailles à raffiner. Elle consiste à représenter la solution à plusieurs niveaux de résolution et à stocker l'erreur entre deux niveaux successifs dans des coefficients dits de *détail*. La différence entre les niveaux de résolution peut-être codée avec des fonctions de base, les ondelettes.

Les méthodes semi-Lagrangiennes peuvent être rendues adaptatives en représentant la fonction de distribution sur un maillage non-uniforme. Plusieurs techniques sont utilisées pour cette représentation : les ondelettes d'interpolation de Deslaurier et Dubuc, aussi appelées "interpolettes" [17]-[3] ou les éléments finis hiérarchiques bi-quadratiques [6].

Nous nous sommes particulièrement intéressés à une méthode utilisant les éléments finis hiérarchiques bi-quadratiques.

Les méthodes par maillages mobiles

Ce sont de méthodes récentes [31] pour lesquelles on considère un maillage uniforme auquel on applique des transformations (rotation, translation) pour "suivre" le mouvement des particules. Ainsi les noeuds du maillage ne correspondent plus aux mêmes points de l'espace des phases mais la valeur de la fonction de distribution en ces noeuds n'a pas besoin d'être mise à jour. On peut aussi par isomorphisme garder un maillage fixe et transformer l'espace des phases.

1.3 Parallélisation des méthodes numériques

Les méthodes numériques sont basées sur un maillage de l'espace des phases, aux noeuds duquel se trouvent les données nécessaires au calcul de la fonction de distribution

[1] les valeurs de la fonction de distribution varient beaucoup

(soit directement les valeurs de la fonction, soit les détails permettant de reconstruire les valeurs). Ce maillage implique naturellement un algorithme *data-parallèle*. En effet, le data-parallélisme (ou parallélisme de données), est adapté lorsque la même opération ou le même traitement est effectué sur toutes les données d'une même structure. On peut alors effectuer en parallèle le traitement sur des données différentes.

1.3.1 Distribution du maillage

La parallélisation de ces méthodes consiste généralement à partitionner le maillage et à attribuer chacune des parties à un processeur. Deux objectifs doivent être atteint pour obtenir des performances optimales :

1. le nombre de noeuds de chaque partie doit être approximativement le même
2. le nombre de noeuds à la frontière de chaque partie doit être aussi petit que possible

Ces deux conditions doivent être vérifiées pour assurer que la charge de travail est bien équilibrée (1) et que le surplus de communication et de calcul est le plus faible possible (2).

Ce problème peut se ramener à un problème de partitionnement de graphe. Pour chaque élément du maillage, on définit un sommet du graphe, et pour chaque pair d'éléments du maillage qui ont une face commune, on définit une arête du graphe. Il faut donc partitionner ce graphe en P sous-graphes de taille identique, où P est le nombre de processeurs. Chacune de ces parties doit être de taille identique et le nombre d'arêtes entre chaque sous-graphe doit être minimum[21, 35]. Or il est bien connu que trouver une solution exacte à ce problème est NP-difficile.

Néanmoins une "bonne" approximation est suffisante, et de nombreuses heuristiques ont été développées et analysées depuis quelques années.

1.3.2 Algorithmes de bisections récursives

En considérant le problème de la division d'un domaine en deux parties, une solution simple est de le couper par une ligne (en $2D$) ou par un plan (en $3D$). Tous les éléments du domaine se trouvant d'un côté de la coupe appartiennent à la première partie, et ceux se trouvant de l'autre côté appartiennent à la deuxième. La taille des parties dépend de l'emplacement de la coupe.

Lors du calcul, des communications doivent être effectuées pour tous les éléments en contact avec cette coupe, on veut donc chercher à effectuer une coupe qui minimise le nombre d'éléments frontière. Selon Cao, Gilbert et Teng [7], si le rapport entre les éléments les plus fins et les plus grossiers du maillage est borné, alors une ligne ou un

plan permettent de garder cette quantité petite. De nombreux algorithmes se basent sur cette idée.

Recursive Coordinate Bisection

Introduit par Berger et Bokhari [2], cet algorithme utilise des coupes (plans ou lignes) qui sont orthogonales aux axes de coordonnées du domaine. On regarde selon quelle coordonnée la répartition des objets est la plus étirée et on divise le domaine en deux parties contenant un même nombre d'objets chacune. On applique ensuite récursivement le même opération sur les deux moitiés générées. On est donc en présence d'un algorithme de type "diviser pour régner". Initialement, on ne peut que créer un nombre d'ensembles de même taille qui soit une puissance de deux, mais la taille des parties peut être ajuster pour en créer un nombre quelconque.

Unbalanced Recursive Bisection

Cet algorithme est présenté comme une amélioration de l'algorithme RCB qui coupe le domaine en deux, ne prenant pas en compte les propriétés géométriques des parties ainsi construites. En effet, le nombre de communications étant corrélé à la taille des bords des parties, il faut éviter de construire des ensembles mal proportionnés, ce qui peut notamment être le cas de la méthode RCB si la densité d'objets de l'ensemble à découper varie beaucoup. La méthode URB évite de créer de telles régions en considérant l'aspect géométrique au moment de choisir une coupe. Pour P processeurs, au lieu de diviser automatiquement le domaine en deux sous-ensembles, on considère les $P-1$ partitions obtenues en formant des sous-ensembles de taille relative i/P et $(P-i)/p$ pour $i \in [1, P-1]$. On sélectionne alors la coupe qui minimise le rapport $longueur/largeur$ de la partition puis on recommence récursivement [22].

Recursive Inertial Bisection

Ici on ne restreint plus les coupes à des lignes ou des plans orthogonaux aux axes. On traite les objets comme des points de masse, dont on calcule l'axe d'inertie principal. Cet axe virtuel est alors coupé orthogonalement de manière à ce que la masse se répartisse équitablement des deux côtés. On fait de même récursivement sur chaque nouvelle partie [11].

Recursive Graph Bisection

Cette technique utilise des informations sur la connectivité pour réduire le nombre de communications engendrées. Le domaine, le plus souvent un maillage, est vu comme

étant un graphe, où chaque sommet représente un point, et chaque arête représente un lien entre deux points voisins. Donc le nombre d'arêtes traversant les frontières des différentes parties représente de possibles communications. Cette méthode utilise la distance sur un graphe : la distance entre deux sommets étant le plus petit nombre d'arêtes à traverser pour aller de l'un à l'autre [30]. L'algorithme revient alors à déterminer les deux extrémités du graphe, à savoir les deux sommets les plus éloignés l'un de l'autre, puis à affecter chacun des sommets restant à la plus proche de ces deux extrémités.

Recursive Spectral Bisection

Cette technique est basée sur la matrice Laplacienne[2] ($L(G)$) du graphe G associée au maillage. Une fois cette matrice établie, on détermine sa 2^e plus grande valeur propre, et le vecteur propre associé. On peut alors trier les sommets (donc les noeuds du maillage) selon la taille de leur entrée dans le vecteur propre, et assigner la moitié à chaque sous-domaine [30, 35]. Puis on recommence récursivement sur chaque domaine ainsi créé.

Les méthodes par bisections récursives permettent d'effectuer de bonnes partitions du maillage mais sont en général assez coûteuses car basées sur une représentation du maillage par un graphe.

1.3.3 Arbres et courbes de recouvrement de surface

Contrairement aux méthodes utilisant des bisections successives, on se base ici sur une division simultanée de chaque axe de coordonnée. On produit ainsi quatre sous-parties en $2D$, et huit en $3D$. On garde la relation entre ces régions au moyen d'un arbre à 4 ou 8 branches. La racine d'un tel arbre représentera donc l'ensemble du domaine.

La traversée de l'arbre définit un ordre sur les feuilles, qui sont les objets du domaine. Cet ordre correspond en général à celui d'une courbe de recouvrement de surface. On peut alors découper cette liste ordonnée autant de fois que nécessaire pour générer n'importe quel nombre de parties [15, 27, 37].

Les domaines construits par les courbes de remplissage ont des frontières plus étendues que les domaines créés par les méthodes basées sur des bisections récursives, mais cette méthode à l'avantage d'être rapide et incrémentale[3].

[2] $L(G) = -D + A$ où D est la matrice diagonale des degrés des sommets de G et A est la matrice d'adjacence de G (i.e $a_{ij} = 1$ si il existe une arête allant des sommets i à j du graphe)

[3] les modifications des domaines sont proportionnelles aux changements du maillage

1.3.4 Améliorations locales

Les techniques vues précédemment sont relativement coûteuses puisqu'elles se basent sur une connaissance globale de l'état du calcul. Les méthodes par améliorations locales proposent une alternative peu coûteuse, en visant à atteindre l'équilibre entre processeurs voisins uniquement. La charge n'est comparée qu'avec les processeurs voisins, et les objets qui sont migrés en cas de déséquilibre ne le sont qu'entre voisins.

Les méthodes par améliorations locales s'apparentent à des algorithmes de diffusion, qui peuvent se baser tout naturellement sur la partition existante à faible coût. Mais si on veut atteindre l'équilibrage global il faut recommencer l'algorithme plusieurs fois, ce qui peut poser des problèmes de convergence.

1.3.5 Équilibrage de charge dynamique

De nombreuses versions parallèles de méthodes uniformes ont été développées. Le maillage étant constant, ces méthodes sont facilement parallélisables en partageant la charge de calcul ainsi que de la quantité importante de mémoire nécessaire pour stocker tous les points du maillage entre les différents processeurs [34].

Avec les méthodes adaptatives, si le maillage, une fois partitionné, est modifier par l'addition de nouveaux éléments ou le retrait d'éléments existants, alors un déséquilibre de charge est immédiatement créé. Il convient alors de déterminer un algorithme d'équilibrage de charge dynamique, qui devra être rapide et se baser si possible sur les partitions déjà existantes.

Chapitre 2

Méthode de résolution

Nos travaux portent sur la parallélisation d'une méthode numérique. Cette méthode est une méthode adaptative, basée sur le schéma semi-Lagragien et les éléments finis hiérarchiques. Cette méthode est décrite dans [6]. Dans un premier temps nous présenterons les grandes lignes de cette méthode. Puis nous donnerons les définitions mathématiques sous-jacentes. La méthode est présentée dans le cas $2D$ et les définitions se généralisent facilement à des dimensions supérieures.

2.1 Principe général

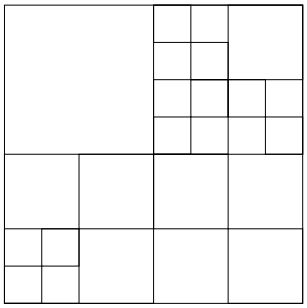


FIG. 2.1 – maillage multi-résolution dyadique

Le maillage utilisé est un maillage *dyadique*, c'est-à-dire une subdivision en base 2 de l'espace des phases (FIG 2.1). On note $\mathcal{M}^n$ le maillage à l'instant $t^n = n\Delta t$.

Chaque subdivision du maillage est appelée *cellule*. Le niveau d'une cellule α est le nombre de subdivisions successives nécessaires pour la construire, et est noté $|\alpha|$. Ainsi une cellule de niveau j, correspond à une cellule résultat de la partition du domaine par

2^{2j} cellules carrées identiques. On note $\mathcal{N}(\mathcal{M}^n)$ l'ensemble des noeuds du maillage $\mathcal{M}^n$, on a 9 noeuds par cellule.

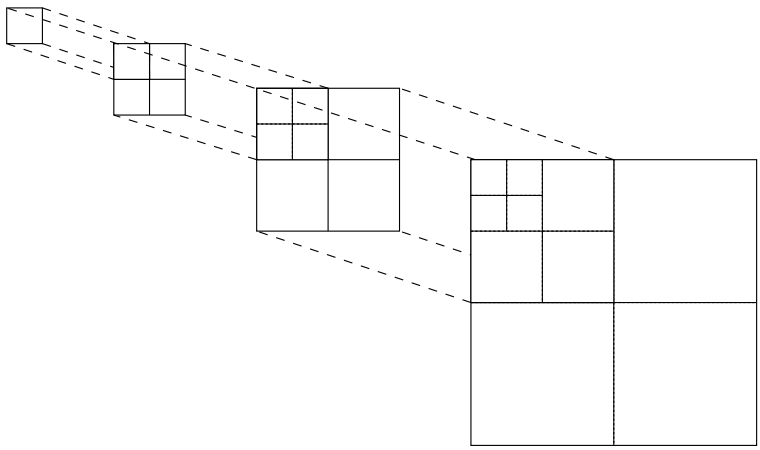


FIG. 2.2 – consistance du maillage

La subdivision d'une cellule de niveau j ne peut s'effectuer qu'en la remplaçant par quatre cellules *filles*[1] de niveau $j+1$. Cette subdivision est appelée *raffinement* et un maillage qui respecte ces propriétés est appelé *consistant* (FIG 2.2). Aux noeuds a de ce maillage est représentée la valeur de la fonction de distribution, notée $f(a)$.

On appelle *courbes caractéristiques* de l'équation de Vlasov, les courbes qui décrivent le mouvement des particules de l'instant t^n à t^{n+1}. Suivant le principe de base de la méthode, la valeur du noeud a, discrétisant la fonction de distribution pour un couple (x, v) à l'instant t^{n+1}, peut être retrouvée en remontant le long de la courbe caractéristique jusqu'à l'instant t^n.

On appelle *advection avant* l'opération qui permet de calculer le point de la courbe à l'instant t^{n+1} à partir du point de cette courbe à l'instant t^n. Inversement, l'*advection arrière* permet de retrouver de calculer le point de la courbe à l'instant t^n à partir du point de cette courbe à l'instant t^{n+1}. Ces deux opérations définissent une bijection entre les points de l'espace des phases.

[1]la cellule de niveau j est appelée cellule *mère*

2.1.1 Les différentes phases de l'algorithme

Le but de cet algorithme est de donner la solution de l'équation à l'instant t^{n+1} à partir de celle à l'instant t^n. On veut donc construire le maillage $\mathcal{M}^{n+1}$ à partir de $\mathcal{M}^n$.

L'algorithme comprend trois phases successives : la prédiction du nouveau maillage, l'évaluation de la fonction de distribution en chaque noeud du nouveau maillage et la compression du maillage.

1. **La prédiction du nouveau maillage**

   On prend le centre de chaque cellule de $\mathcal{M}^n$. Pour chacun de ces centres, on détermine le point résultat de son advection en avant (FIG 2.3). On considère ce point dans le maillage $\mathcal{M}^{n+1}$.

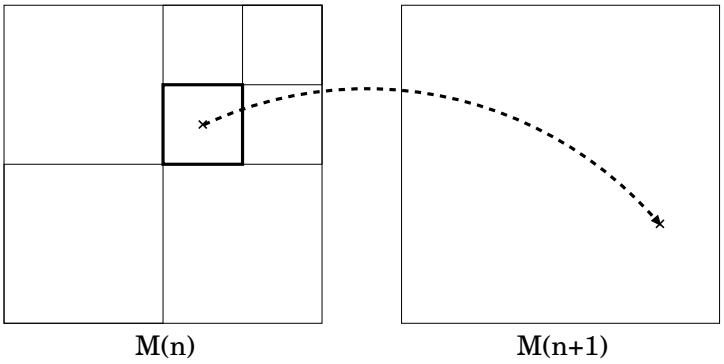


FIG. 2.3 – advection en avant du centre de la cellule

   On construit alors la cellule de $\mathcal{M}^{n+1}$, qui soit de même niveau qui contienne ce point. Cette cellule est unique (FIG 2.4).

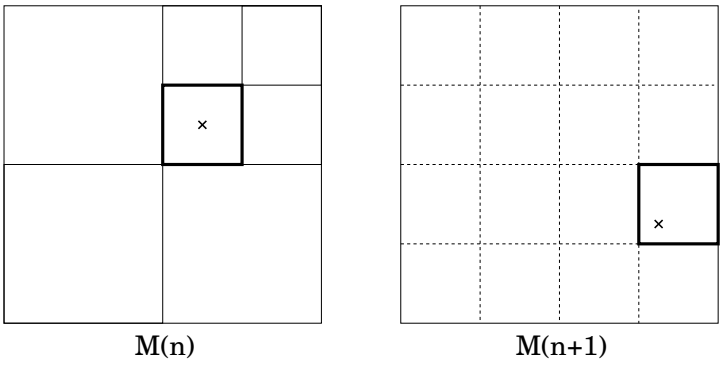


FIG. 2.4 – construction de la cellule prédite

Puis on construit les cellules nécessaires pour garantir la consistance du maillage et on raffine la cellule issue de la prédiction d'un niveau. Ce raffinement nous permet d'adapter la solution à la simulation dans le cas où le gradient serait important dans cette partie du maillage (FIG 2.5).

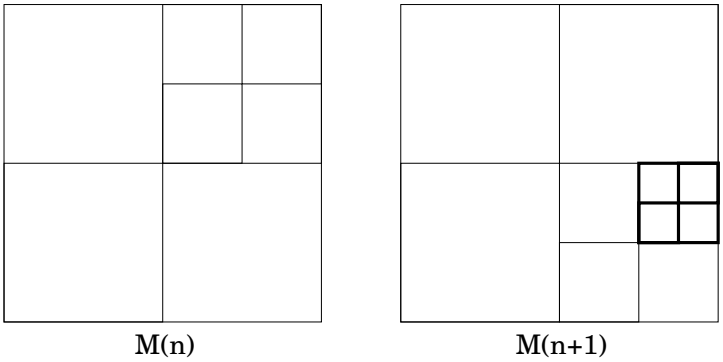


FIG. 2.5 – ajout de cellules : consistance et raffinement

2. **L'évaluation de la fonction de distribution**

Une fois le nouveau maillage prédit, on veut retrouver les valeurs de la fonction de distribution à l'étape $n+1$. Donc pour chaque noeud a de $\mathcal{M}^{n+1}$, on calcul le point résultat de son advection en arrière (FIG 2.6). On considère ce point sur le maillage $\mathcal{M}^n$.

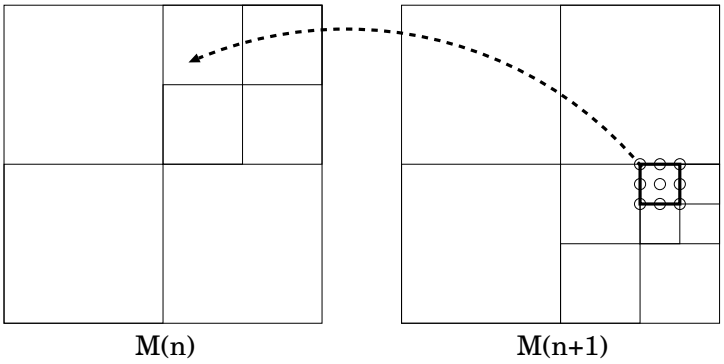


FIG. 2.6 – advection en arrière

Puis on interpole la valeur en ce point à l'aide des noeuds de la cellule qui le contient dans $\mathcal{M}^n$. Enfin on affecte cette valeur au noeud a de $\mathcal{M}^{n+1}$ (FIG 2.7).

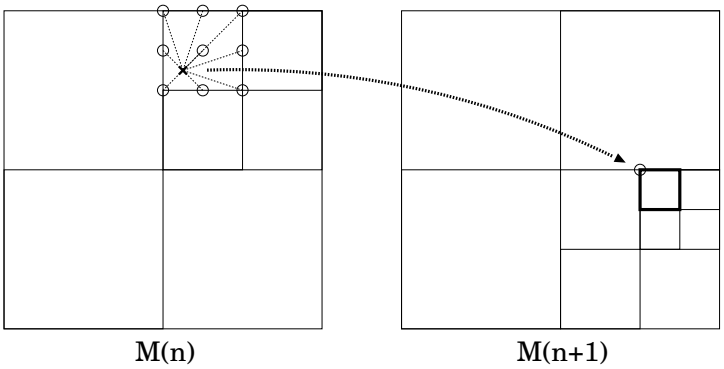


FIG. 2.7 – interpolation et rapatriement de la valeur

3. **La compression du maillage**

Enfin pour permettre une diminution des cellules dans les zones à faible gradient, pour chaque groupe de quatre cellules filles issues de la même mère, on interpole la valeur en leurs noeuds par ceux de la cellule mère (noeuds en noir dans la FIG 2.8).

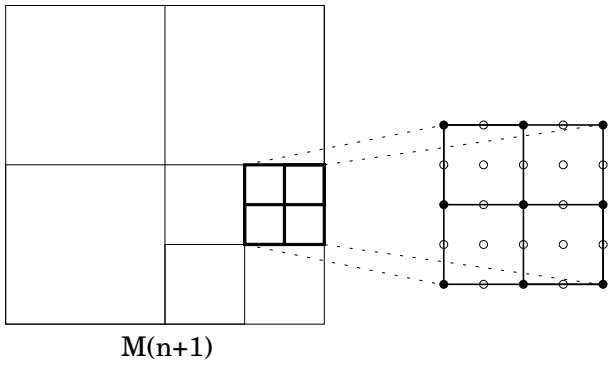


FIG. 2.8 – test de compression

Si pour tous les noeuds des cellules filles, la différence entre la valeur au noeud et

la valeur interpolée est inférieure à un certain seuil, alors on remplace les quatre cellules filles par la cellule mère (FIG 2.9).

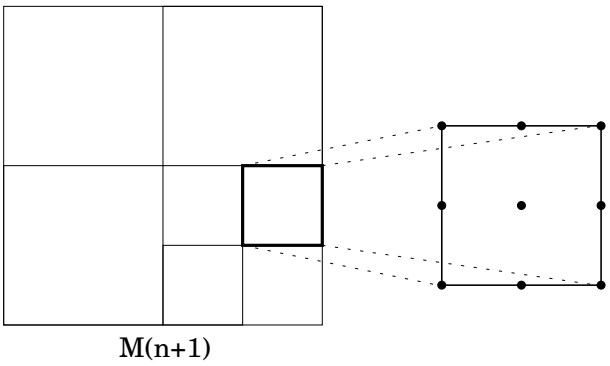


FIG. 2.9 – élimination de cellule

2.2 Description mathématique

2.2.1 Le maillage adaptatif dyadique

En considérant le domaine de calcul comme étant le carré unité $[0,1]^2$, on note $\mathcal{M}_j$ l'ensemble de toutes les cellules de résolution $j \in \mathbb{N}$ de l'espace des phases dyadique :

$$\mathcal{M}_j := \{\alpha_{k,l}^j := [k\,2^{-j}, (k+1)\,2^{-j}] \times [l\,2^{-j}, (l+1)\,2^{-j}] \ : \ k,\, l \in \mathbb{Z}\,\},$$

Pour gagner en flexibilité dans la discrétisation des solutions numériques, on construit un maillage de résolution variable, et une manière naturelle de le faire est de définir un *arbre de cellules*. Pour cela on décrit les enfants et parents d'une cellule α par :

$$\mathcal{D}(\alpha) := \{\ \beta \in \mathcal{M}_{|\alpha|+1} \ : \ \beta \subset \alpha\ \}$$

et

$$\mathcal{P}(\alpha) := \{\ \beta \in \cup_{j \le |\alpha|-1} \mathcal{M}_j \ : \ \beta \supset \alpha\ \},$$

où $|\alpha|$ est le *niveau* logique de la cellule α.

On définit alors un **arbre adaptatif** Λ comme un ensemble de cellules satisfaisant les deux propriétés suivantes :

1. $\mathcal{M}_{j_0} \subset \Lambda$
2. $\forall \alpha \in \Lambda,\ \bigcup_{\beta \in \mathcal{P}(\alpha)} \mathcal{D}(\beta) \subset \Lambda.$

On appelle raffinement d'une cellule le fait de la remplacer par ses filles. La propriété 2 implique alors qu'il ne peut pas y avoir de cellule de Λ qui soit *partiellement* raffinée, donc les feuilles (*i.e.* les cellules non raffinées) forment une partition de l'espace des phases. On appelle un tel ensemble $\mathcal{M} = \mathcal{M}(\Lambda)$ un **maillage adaptatif consistant**.

2.2.2 La représentation bi-quadratique adaptative

On note $\Gamma^2(\alpha)$ et on appelle noeuds, les 9 points équidistants d'une cellule bi-quadratique α, et on note $\Gamma^2(\mathcal{M})$ les noeuds de toutes les cellules d'un maillage adaptatif $\mathcal{M}$ donné.

Alors la solution numérique à un temps t^n est la donnée :

$$\mathcal{F}_n := (\mathcal{M}^n, \quad (f^n(a))_{a\in\Gamma^2(\mathcal{M}^n)}), \tag{2.1}$$

où $\mathcal{M}^n$ est le maillage adaptatif au temps t^n, et a est un noeud de ce maillage.

L'évaluation $f^n(c)$ de la solution à un point $c \in [0,1]^2$ est obtenue en recherchant l'unique cellule α du maillage adaptatif $\mathcal{M}^n$ où ce point est présent, en utilisant les valeurs $f^n(\alpha) := (f^n(a))_{a\in\Gamma^2(\alpha)}$ et en calculant l'interpolation bi-quadratique locale pour cette cellule, que l'on note $I(c, \alpha, f^n(\alpha))$.

2.2.3 L'opération d'advection et les caractéristiques

On note $(X, V)(s; x, v, t)$ les caractéristiques de l'équation de Vlasov, i.e. les solutions du système d'équations différentielles suivant :

$$\begin{cases} \dot{X}(s) = V(s) \\ \dot{V}(s) = E(s, X(s)) \end{cases}$$

avec les conditions initiales $X(t) = x$ et $V(t) = v$.

En connaissant la position finale $a = (X^{n+1}, V^{n+1})$ au temps t^{n+1}, on définit le noeud advecté en arrière $B_n^{wd}(a) := (X^n, V^n)$ comme une approximation du second ordre de sa position arrière exacte $(X, V)(t^n; a, t^{n+1})$.

On définit de manière analogue l'advecté en avant d'un noeud $a = (X^n, V^n)$ par $F_n^{wd}(a) := (X^{n+1}, V^{n+1})$.

2.2.4 L'algorithme détaillé

▷ **Initialisation**

La fonction initiale est donnée par f_0 sur le carré unité, et le maillage $\mathcal{M}^0$ est vide.

• calculer $d(b) := f_0(b) - I(b, \alpha, f_0(\alpha))$ pour $b \in \Gamma^2(\mathcal{D}(\alpha))\backslash\Gamma^2(\alpha)$

• ajouter α dans $\mathcal{M}^0$ si le test de compression suivant est faux :

$$\sum_{b\in\Gamma^2(\mathcal{D}(\alpha))\backslash\Gamma^2(\alpha)} \omega_{|\alpha|}|d(b)| \leq \varepsilon, \tag{2.2}$$

où $|\alpha|$ est le niveau de la cellule α, ω_j prend pour valeur 1, 2^{-dj} ou $2^{-dj/2}$ suivant la norme sur laquelle on se base pour la compression (respectivement L_∞, L_1 ou L_2), et d est la dimension du maillage. Dans la simulation traitée ici, on prend toujours la norme L_2 avec $d = 2$.

• ajouter les cellules nécessaires pour avoir un maillage adaptatif consistant

Dans la suite de l'algorithme on boucle sur le temps n.

▷ **Prédiction de $\widetilde{\mathcal{M}}^{n+1}$**

Le nouveau maillage $\widetilde{\mathcal{M}}^{n+1}$ est vide au départ. Pour chaque centre c des cellules α du maillage $\mathcal{M}^n$:

• calculer les points advectés en avant $F_n^{wd}(c)$ (FIG 2.3)

• ajouter l'unique cellule de niveau $|\alpha|$ qui se trouve à cette place dans $\widetilde{\mathcal{M}}^{n+1}$ (FIG 2.4)

• rajouter les cellules nécessaires pour conserver la consistance de $\widetilde{\mathcal{M}}^{n+1}$

• raffiner d'un niveau les cellules (qui ne sont pas de niveau le plus fin J), autrement dit remplacer chaque cellule par ses filles dans l'arbre adaptatif (FIG 2.5)

▷ **Advection semi-Lagrangienne**

Pour chaque noeud $a \in \Gamma^2(\widetilde{\mathcal{M}}^{n+1})$:

- calculer le point advecté en arrière (FIG 2.6)
- mettre la valeur de $\widetilde{f}^{n+1}(a)$ à $f^n(B_n^{wd}(a))$ (FIG 2.7)

▷ **Compression du maillage**

On a donc une première fonction de distribution $\widetilde{\mathcal{F}}_{n+1}$ que l'on veut compresser pour obtenir $\mathcal{F}_{n+1}$. De $j = J-1$ à j_0+1, pour chaque cellule $\alpha \in \widetilde{\mathcal{M}}^{n+1}$ de niveau j :

- calculer $d(b) := \widetilde{f}^{n+1}(b) - I(b, \alpha, \widetilde{f}^{n+1}(\alpha))$ pour $b \in \Gamma^2(\mathcal{D}(\alpha))\backslash\Gamma^2(\alpha)$
- enlever α de $\widetilde{\mathcal{M}}^{n+1}$ si le test de compression (2.2) est vrai

Il reste à la fin $\mathcal{F}_{n+1}$.

Chapitre 3

Parallélisation de la méthode semi-Lagrangienne

La méthode numérique induit un algorithme data-parallèle si l'on considère le maillage adaptatif comme une structure de données parallèle. Donc la parallélisation revient à distribuer le maillage (cellules et noeuds) entre les processeurs.

Nous avons conçu deux versions de l'algorithme : la première en distribuant uniquement les noeuds, pour permettre à notre algorithme de minimiser les communications, la seconde en distribuant les noeuds et les cellules entre tous les processeurs pour augmenter le degré de parallélisme.

Il s'agit du même algorithme data-parallèle, dans lequel une des particularités commune aux deux versions est d'effectuer la compression uniquement sur les cellules locales aux processeurs. Mais la compression peut s'effectuer sur plusieurs niveaux successifs. C'est donc une approximation de la méthode numérique classique, puisque l'on compresse moins de cellules à chaque itération, mais cela ne remet pas en cause la convergence.

On va maintenant détailler pour chacune de ces versions le schéma de distribution du maillage ainsi que la structure de données utilisée et les communications engendrées par ce schéma.

3.1 Version noeuds distribués

On choisit dans cette version de ne distribuer que les noeuds du maillage entre les différents processeurs.

3.1.1 Schéma de distribution

Chaque processeur possède en local l'ensemble des cellules du maillage, dont certaines lui sont spécialement affectées, pour lesquelles il possède les noeuds et leur valeur (FIG 3.1. Il connaît de plus l'identité des processeurs qui possèdent les noeuds des cellules qui ne lui sont pas affectées. On reprend donc l'algorithme séquentiel de la méthode numérique et on distribue le calcul de l'interpolation des valeurs lors de la phase d'évaluation (ou d'advection semi-Lagrangienne), ainsi que l'ensemble de la phase de compression.

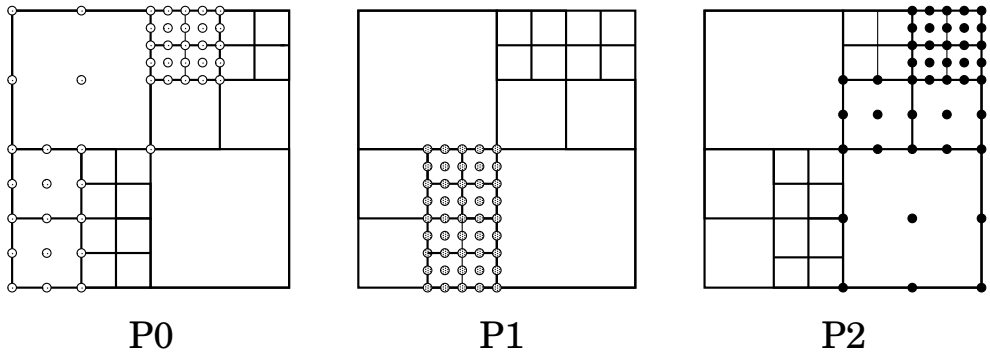


FIG. 3.1 – distribution des noeuds

3.1.2 Structures de données

Le maillage est représenté par des tables de hachage pour des raisons d'efficacité, on peut ainsi garantir un accès en temps constant aux éléments du maillage. Chaque processeur dispose en local :

- d'une table pour les cellules avec en entrée l'index global de la cellule et en sortie le couple cellule - identifiant du processeur d'affectation
- d'une table pour les noeuds avec en entrée l'index global du noeud et en sortie le couple noeud - valeur de la fonction de distribution en ce point du domaine

Comme la méthode semi-Lagrangienne se base sur les valeurs à l'étape de temps précédente pour calculer celles de l'étape actuelle, chaque processeur possède ce couple de table représentant sa vision du maillage pour deux étapes de temps successives.

Contrairement à la méthode numérique et au code séquentiel, on décide de conserver dans la table des cellules uniquement les cellules qui sont les feuilles de l'arbre adaptatif, pour minimiser l'usage mémoire qui est l'un des enjeux principaux pour le passage en dimensions supérieures.

3.1.3 Minimisation des communications

Un schéma de communication spécifique, pour recouvrir le temps des communications par des calculs [34], est utilisé pour minimiser la perte de temps qu'elles génèrent. On utilise pour cela des fonctions d'envoi et de réception non bloquantes, initialisées le plus tôt possible.

Avec cet algorithme, la majorité des communications sont effectuées lors la phase d'advection. Il faut donc chercher à les minimiser au maximum lors de cette phase sans changer l'architecture du programme.

Le principe est, lors de la phase d'advection, de changer le propriétaire de la cellule α du maillage $\widetilde{\mathcal{M}}^{n+1}$, si les advectés en arrière de tous ses noeuds se retrouvent dans le maillage $\mathcal{M}^n$ sur un autre unique processeur. Ainsi les calculs d'interpolation pour déterminer la valeur des noeuds s'effectueront en local et on diminuera le nombre de communications effectuées.

Le problème est alors que les cellules appartenant à un même processeur sont réparties de manière plus hétérogène sur le maillage, ce qui provoque une forte baisse du taux de compression des cellules. En effet la compression étant uniquement effectuée en local sur un processeur, celui-ci ne peut éliminer les cellules dont il ne possède pas toutes les soeurs. Il en résulte donc un accroissement important du nombre total de cellules dans le maillage, ce qui provoque une baisse des performances globales de la simulation.

3.2 Version noeuds et cellules distribuées

On choisit ici de distribuer l'ensemble du maillage, cellules et noeuds, pour offrir un degré de parallélisme supérieur. En effet une trop grande partie du code était précédemment traitée de manière séquentielle, ne tirant pas assez profit de l'architecture parallèle.

3.2.1 Schéma de distribution

On appelle *région* une surface du domaine de calcul, définie par une union de cellules du maillage. On divise le domaine de calcul en *régions*, l'ensemble de ces régions formant une partition du domaine, et on alloue chacune d'elle à un processeur. Un processeur possède alors toutes les cellules et points contenus dans la région qui lui est affectée (FIG 3.2).

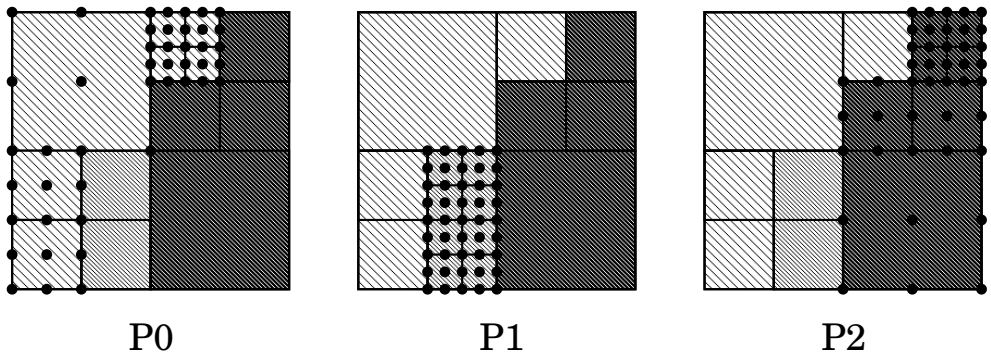


FIG. 3.2 – distribution des *régions*

3.2.2 Structure de données

On utilise les même structures de données que dans la section 3.1.2, à la différence près que la table des cellules d'un processeur ne contient pas toutes les cellules du maillage, mais uniquement les cellules de sa région, qui forment une partition de cette dernière. Un processeur possède bien sur, dans sa table des noeuds, tous les noeuds et valeurs des cellules de sa région.

De plus pour éviter des communications superflues, on conserve des cellules appartenant aux autres processeurs, celles-ci forment la partition la plus grossière des régions d'où elles sont issues. Ainsi les cellules contenues dans une table forment toujours une partition de l'ensemble du domaine physique. L'identité du processeur possédant n'importe

quelle cellule du maillage peut donc toujours être retrouver grâce à cette information. Cette propriété est surtout utilisée lors des advections avant et arrière pour déterminer la région d'appartenance du point solution (donc l'identifiant de processeur).

3.2.3 Communications

Un processeur n'ayant qu'une vision locale du maillage ne connaît que les cellules et noeuds de sa propre région. Cette implantation nécessite donc plus de communications que la version précédente Ce surplus est dû notamment à l'ajout de cellules lors de la phase de prédiction et à l'évaluation des valeurs aux noeuds lors de la phase d'advection semi-Lagrangienne.

Détail du schéma de communications pour chaque phase :

Prédiction

Chaque processeur réalise cette phase pour toutes les cellules de sa région. Un processeur p advecte en avant le centre c_α d'une cellule α et créé la cellule $\bar{\alpha}$ de même niveau contenant le point $\bar{c_\alpha}$ résultat de l'advection. Si $\bar{\alpha}$ se trouve dans la région d'un autre processeur p', alors elle lui est envoyée et c'est p' qui assurera son insertion dans le maillage (dans sa région locale) en y respectant la consistance. Un problème pour les communications est que le nombre de communications à effectuer et l'identité des processeurs émetteurs et récepteurs ne sont pas connus à l'avance. Donc au début de cette phase, une réception de cellule par processeur distant est initialisée sur chaque processeur. Chaque fois qu'une nouvelle cellule arrive sur p en provenance de p', une nouvelle réception pour p' est initialisée. Un message spécifique *end-of-send* est envoyé par p à tous les autres processeurs quand il a fini le traitement de toutes ses cellules locales. Quand les autres processeurs recevront ce message ils arrêteront d'initier des réceptions pour p.

Évaluation

Chaque processeur réalise cette phase pour tous les noeuds qu'il possède dans sa région. Si le point $\bar{a}$, résultat de l'advection en arrière du noeud a appartenant à p, n'est pas contenu dans la région de p alors le processeur ne possède pas en local les noeuds et leur valeur pour pouvoir effectuer l'interpolation. $\bar{a}$ est alors envoyée au propriétaire p' de la cellule (donc de la région) où il se trouve et c'est p' qui effectue le calcul de la valeur et la renvoit à p. Le schéma de communication est le même que dans la phase de prédiction, à la différence prêt qu'une réception explicite est initialisée lors de l'envoie du point advecté, pour en récupérer la valeur par la suite.

Compression

Pour cette phase, chaque processeur ne cherche à éliminer que ses cellules locales, celles qui appartiennent à sa région. On évite ainsi toute communication entre les processeurs. C'est une approximation de la méthode numérique parce que moins de cellules sont éliminées, mais cela ne remet pas en cause la convergence.

Chapitre 4

Stratégie d'équilibrage proposée

On propose ici un mécanisme d'équilibrage de charge pour la version avec distribution de l'ensemble du maillage présentée en section 3.2. En effet, le maillage s'adaptant dans le temps à l'évolution du système physique, le nombre de cellules (donc le nombre de noeuds) à l'intérieur d'une même région varie. De plus, après les phases de prédiction et d'advection, tous les processeurs sont synchronisés à cause des communications. Cela implique des temps d'attente entre processeurs. Il devient donc nécessaire d'inclure un mécanisme d'équilibrage de charge dynamique efficace dans notre algorithme, qui consistera à redéfinir les régions au cours de l'exécution.

Dans un premier, il faut définir les critères pour former des régions qui correspondent à nos attentes. On proposera ensuite des méthodes pour satisfaire ces critères, puis on présentera un mécanisme les mettant en oeuvre.

4.1 Critères des régions

Nous avons deux objectifs : avoir des régions bien équilibrées pour minimiser les temps de synchronisation et avoir des régions bien formées pour minimiser les communications et améliorer la compression.

4.1.1 Équilibre des régions

La première caractéristique de nos régions est d'avoir une quantité de charge approximativement identique. Il faut donc modéliser cette charge pour notre algorithme.

Le problème ici est que le code considéré est découpé en trois parties distinctes, et que chacune de ces parties a des incidences différentes sur la charge de calcul. Trois traitements différents sont possibles [18] :

1. déterminer quelle est la phase la plus coûteuse du code, et concentrer l'algorithme d'équilibrage de charge sur cette phase. Cette méthode peut être efficace dans le cas d'un fort déséquilibre entre les charges respectives des différentes phases, ce qui n'est pas le cas de la simulation dont on se préoccupe.
2. implanter un mécanisme d'équilibrage de charge spécifique à chaque phase, pour traiter chacune d'elles de manière optimale. Mais pour que cela soit efficace il faut que le coût d'un tel équilibrage soit minime par rapport aux calculs effectués dans chacune de ces phases. Or dans notre simulation, les calculs sont plutôt élémentaires mais la quantité de données est considérable, rendant le traitement de l'équilibrage trop coûteux.
3. effectuer un équilibrage de charge commun aux différentes phases. Celui-ci sera moins performant pour chacune des phases car moins spécialisé, mais sera beaucoup moins coûteux que la méthode précédente. Il nécessite cependant de trouver une modélisation commune de la charge pour les différentes phases.

Étant donné que le coût de la simulation concernées est plus basé sur le nombre important d'opération que sur leur complexité et étant donné que l'algorithme adaptatif à pour but de réduire ce nombre, on choisit de suivre la 3^{e} solution.

L'indicateur de charge des processeurs que nous retenons est le nombre de cellules de chaque région. En effet, cette caractéristique de la charge est commune aux trois phases, c'est donc un bon compromis pour mesurer la charge du système sur une itération entière.

4.1.2 Forme des régions

Comme nous l'avons vu en section 1.3.1, pour minimiser le surplus de communications générées par la décomposition du maillage et les dépendances inter-processeurs, il convient de minimiser le nombre d'éléments frontières de chaque région. Il s'agit donc d'assurer certaines propriétés de forme pour nos régions.

Il est évident que sur le type de maillage que nous considérons, la meilleure forme possible pour une région est le carré. En effet cette forme nous permet de rendre minimale le nombre d'éléments frontière pour un nombre de cellules fixé. Bien sûr la forme en carré implique que la région soit connexe, autrement dit que l'on peut rejoindre n'importe quelle cellule de la région en ne se déplaçant que par des cellules de la même région ayant un bord en commun[1].

Les régions carrées et connexes nous permettent en plus de réduire le défaut de compression dû à l'approximation de la méthode parallèle (voir section 3). En effet, de telles régions conservent la majeure partie de la hiérarchie des cellules, ce qui peut permettre de compresser sur plusieurs niveaux si le gradient est très faible.

[1] i.e elles ont au moins deux noeuds en commun

On a donc trois critères pour nos régions : elles doivent être connexes, "carrées" et composées approximativement du même nombre de cellules,

4.2 Outils mathématiques

L'objectif est de satisfaire au mieux les critères de forme et de connexité par des outils spécifiques et de la manière la moins coûteuse possible.

4.2.1 Quad-arbre

Pour modéliser cette notion de hiérarchie de cellules, le mailage adaptatif est représenté par un 4-arbre, ou arbre à quatre branches. Cette structure correspond naturellement au maillage dyadique où chaque cellule de niveau j possède quatre cellules filles de niveau $j+1$.

Les cellules du maillage sont représentées par les feuilles de l'arbre. La hiérarchie de cellules, donc les parents "virtuels" des cellules existantes à un temps n, est représentée par les noeuds de l'arbre. Le nombre de feuilles de chaque les sous-arbre est représenté à sa racine. On représente ainsi, par chaque noeud du quad-arbre, des régions du maillages et la charge de calcul qu'elles représentent. Ainsi la charge peut être facilement localisée.

Pour construire l'arbre, on se base uniquement sur l'index global des cellules. On trouve le point d'ancrage de la cellule α à insérer, en prenant à chaque noeud parcouru la branche qui mène à la cellule fille contenant α. La méthode utilisée pour retrouver l'index de cette cellule fille la méthode suivante :

Soit (x', v') l'index de cellule de niveau l, $l < |\alpha|$, qui contient α dont l'index local est (x, v). Alors on a :

$$x' = \left\lfloor \frac{x}{2^{|\alpha|-l}} \right\rfloor, v' = \left\lfloor \frac{v}{2^{|\alpha|-l}} \right\rfloor \tag{4.1}$$

L'index relatif de la branche de l'arbre est alors obtenu en *mappant* l'index relatif de la cellule par rapport à sa mère $(x' mod\ 2, v' mod\ 2)$ dans un tableau $1D$. On utilisera ici l'*état* local de la courbe de Hilbert, défini dans la prochaine section, pour établir cette correspondance.

4.2.2 Courbe de remplissage de surface

Pour assurer la connectivité des régions, il faut définir une relation de voisinage entre les cellules. Dans le cas d'un maillage adaptatif, on ne peut définir facilement les 4 ou les 8-voisins, chaque cellule ayant un nombre variable de voisins. Pour établir une relation de

voisinage on ordonne les cellules par une courbe de remplissage de surface[2] (*Space Filling Curve*). On choisit la courbe de Hilbert [25], qui a plusieurs avantages :

1. elle conserve la connexité
2. elle est définie sur N-dimensions
3. elle peut être construite sur des maillages multi-résolution (FIG B.1,B.2)

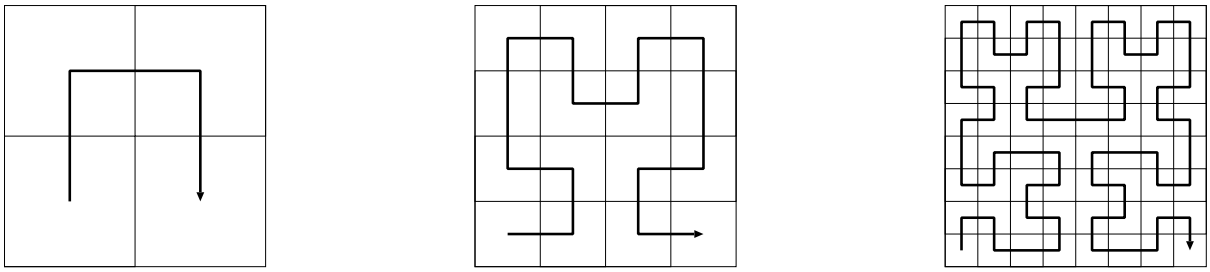


FIG. 4.1 – courbe de Hilbert du 1er, 2e et 3e ordre

On utilise le diagramme de Lawder et King [23] (FIG 4.2) pour construire la courbe de Hilbert dans le cas $2D$.

Dans ce diagramme donné en FIG 4.2) les chiffres entre parenthèses sont les coordonnées relatives des cellules filles. Un état correspond à une configuration de la courbe de Hilbert. Pour connaître depuis un état donné, l'état d'une des cellules filles, il suffit alors d'en connaître les coordonnées relatives et de suivre la flèche.

Comme l'algorithme pour calculer cette courbe correspondant au maillage adaptatif est récursif, elle peut être calculée en même temps que la construction de l'arbre. Les branches de cet arbre sont alors ordonnées selon *l'état* de la courbe au noeud courant (voir section 4.2.1).

Un état correspond à une configuration de la courbe de Hilbert, donc donne un ordre relatif pour quatre cellules soeurs. L'ordre pour les cellules filles d'une cellule mère se trouvant dans l'état 0 sera donc : $(0,0) \rightarrow (0,1) \rightarrow (1,1) \rightarrow (1,0)$.

Ainsi un simple parcours de l'arbre en *profondeur d'abord* correspondra au parcours selon la courbe de Hilbert. On peut maintenant *découper* simplement l'arbre pour générer des partitions connexes de cellules, il ne reste donc plus qu'à déterminer les emplacements de ces coupes pour que ces partitions soient équilibrées et surtout qu'elles aient une bonne forme.

[2]courbe continue remplissant un carré

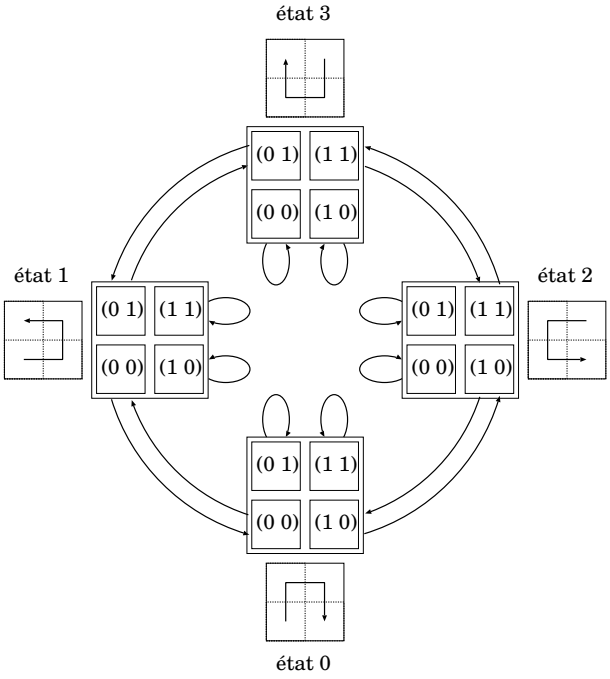


FIG. 4.2 – diagramme de calcul de la prochaine configuration de la courbe de Hilbert

4.2.3 Choix du déclenchement de l'équilibrage

On choisit d'effectuer cet équilibrage entre les phases de prédiction et d'advection, pour assurer que l'advection et la compression (les deux phases les plus coûteuses) s'effectuent sur des régions équilibrées. De plus cela permet de ne changer que l'affectation des cellules du maillage temporaire prédit, on ne transmet aucun noeud.

La détection du déséquilibre est basée sur l'état de charge global du système : chaque processeur envoie à tous les autres sa charge locale après prédiction. Ainsi chaque processeur peut calculer la charge globale totale du système, la charge idéale et la plus grande différence de charge entre deux processeurs. Si cette plus grande différence dépasse un certain seuil, fixé par l'utilisateur selon l'importance donnée à la perte de temps des synchronisations, alors on effectue la mise à jour des régions.

4.3 Implantation du mécanisme

On présente deux mécanismes complémentaires. Le premier, permettant de construire les régions, est plutôt utilisé comme partitionneur statique du maillage (en raison de

son coût). L'autre, permet de mettre à jour ces régions de manière dynamique durant l'exécution de la simulation.

4.3.1 Création des régions

L'initialisation peut donner une très mauvaise partition du maillage. Dans un premier temps, l'équilibrage effectué repose sur la connaissance globale du maillage et ne tient pas compte de la précédente répartition des cellules. Le but de cet algorithme est de générer des régions les mieux formées pour ce début de simulation, même si il est coûteux puisque chaque processeur aura à ce moment la connaissance du maillage dans son ensemble.

Charge idéale et facteur d'erreur

La charge idéale $\mathcal{I}$ est la quantité de charge vers laquelle doivent tendre tous les processeurs. Il s'agit simplement de la moyenne des charges de tous les processeurs du système.

$$\mathcal{I} = \frac{\sum_{p \in P} L_p}{\|P\|} \tag{4.2}$$

où P est le nombre total de processeurs du système, et L_p est la charge du processeur p. La charge des régions construites sera alors $\mathcal{I} \pm 1$.

Pour générer des régions ayant de bonnes propriétés de forme, on introduit un facteur d'erreur $\varepsilon \in [0, 1]$ assurant un certain degré de liberté lors de leur création. Ainsi la condition de charge pour la création des régions devient :

$$(1 - \varepsilon) * (\mathcal{I} - 1) \leq L_p \leq (1 + \varepsilon) * (\mathcal{I} + 1) \tag{4.3}$$

Cette condition est au centre de l'algorithme de partitionnement suivant.

Algorithme de partitionnement

Soit P le nombre de processeurs de la simulation, p le processeur courant, $node_{cur}$ le noeud courant de l'arbre et $load_{cur}$ la charge courante de la partition :

Init:

$node_{cur}$ est la racine de l'arbre

$load_{cur}$ est la charge de $node_{cur}$

$p = 0$

TANT QUE $p < P$ **FAIRE**

SI $load_{cur} > (1 + \varepsilon) * (\mathcal{I} + 1)$ **ALORS**

- remplacer $node_{cur}$ par son 1^{er}fils
- affecter la charge de $nodecur$ à $load_{cur}$

SINON SI $load_{cur} < (1 - \varepsilon) * (\mathcal{I} - 1)$ **ALORS**

- affecter le sous-arbre de racine $nodecur$ au processeur p
- remplacer $node_{cur}$ par le noeud suivant à droite
- ajouter la charge de $nodecur$ à $load_{cur}$

SINON

($load_{cur}$ satisfait la condition 4.3)

- affecter le sous-arbre de racine $node_{cur}$ au processeur p
- remplacer p par $p + 1$ et $node_{cur}$ par le noeud suivant à droite
- affecter la charge de $node_{cur}$ à $load_{cur}$

FIN SI

FIN TANT QUE

La fonction d'affectation d'un sous-arbre à un processeur est une simple fonction de parcours en profondeur qui change le propriétaire des cellules correspondant aux feuilles de ce sous-arbre.

Les partitions sont affectées arbitrairement aux différents processeurs, ce qui ne pose pas de problème au début de la simulation, mais doit être évité par la suite. De plus, un algorithme aussi global ne peut être performant dans l'optique d'un équilibrage dynamique. Il ne servira donc qu'à partitionner statiquement le maillage au début de la simulation, pour garantir des régions bien formées. Un autre algorithme d'équilibrage plus local pourra alors se baser sur ces régions.

4.3.2 Mise-à-jour des régions

Il est hors de question de reconstruire localement l'ensemble du maillage sur chaque processeur au cours de la simulation, on se contentera donc de mettre à jour localement chaque région. Durant les tests, il s'est avéré que les temps d'attente, dus aux synchronisations lors des phases de communications, pénalisaient beaucoup le temps d'exécution total. De plus, sur notre architecture cible, les communications souffrent d'avantage de

leur quantité que de leur rapidité. Donc pour favoriser un équilibrage le plus exact possible, on décide de conserver la connaissance globale de l'état de charge du système. Les régions qui sont issues de cet équilibrage restent connexes, mais dont leur forme n'est plus garantie.

Cet algorithme est une méthode hybride, puisqu'il se base sur la connaissance globale de la charge du système, mais ne l'équilibre qu'avec une méthode par amélioration locale. Le principe en est le suivant : chaque processeur ne peut recevoir et envoyer des cellules qu'à ses voisins par la courbe de Hilbert, ceci pour conserver la connexité des régions. Contrairement aux méthodes par améliorations locales pures[3], la quantité de cellules à transférer pour atteindre l'équilibrage parfait en une itération est déterminé grâce à la connaissance de la charge pour chaque processeur.

Migration de la charge

La nombre de cellules à transmettre, ou à recevoir, pour atteindre la charge idéale, doit être déterminée localement pour chaque processeur.

Une première idée est de calculer pour chaque processeur p la différence entre la charge locale L_p et la charge idéale $\mathcal{I}$.

$|\mathcal{I} - L_p|$ est le nombre de cellules à :
- envoyer si ce nombre est négatif
- recevoir si il est positif

Mais cette solution ne permet pas d'équilibrer l'ensemble du système. En effet la courbe de Hilbert définit un ordre strict sur les régions. Or il se peut qu'une région ait besoin de plus de cellules que celles qui sont en excès sur ses deux voisins. Donc un processeur doit pouvoir recevoir des cellules venant de tous les autres processeurs du système, même s'il ne communique directement qu'avec ses deux voisins.

Au lieu de chercher à équilibrer chaque région, on raisonne plutôt en terme de différence totale. L'idée, pour un processeur donné, est de prendre sa région comme un pivot, avec de chaque côté, une masse représentant la somme des différences, par rapport à l'équilibrage idéal, des charges des régions *précédentes* et *suivantes*. Cela détermine précisément la quantité de charge à transmettre de chaque côté pour que la différence par rapport à la charge idéale soit nulle, même si la répartition locale des cellules à envoyer n'est pas connue.

Donc pour chaque processeur $p \in P$ on calcul la différence de charge *gauche* D_g pour

[3]elles se basent uniquement sur la connaissance de la charge sur un voisinage, ce qui implique un phénomène de convergence plus ou moins lente pour atteindre l'équilibrage global du système

l'ensemble des régions précédentes, et *droite* D_d pour l'ensemble des régions suivantes :

$$D_g = \sum_{i=0}^{p-1} \mathcal{I} - L_i \qquad D_d = \sum_{i=p+1}^{P} \mathcal{I} - L_i \tag{4.4}$$

Si $D_g < 0$ (resp. $D_d < 0$) le processeur p envoie ses $|D_g|$ premières cellules (resp. $|D_d|$ dernières cellules) au processeur $p-1$ (resp. $p+1$).

Anisi la forme "carrée" des régions n'est plus explicitement conservée. Mais la courbe de Hilbert préserve la localité, donc la forme générale reste identique même si elle présente localement des irrégularités.

Chapitre 5

Tests et résultats

Le programme développé se base sur le code séquentiel YODA[1] de Michel Mehrenberger et Martin Campos-Pinto. Il est composé de 10000 lignes de code en C++, utilisant la bibliothèque de passage de message MPI. Les tables de hachage sont des hashmap de la STL, permettant la parcours de la table par itérateurs. Le code est évolutif, de nombreuses *templates* permettent de paramétrer les différentes dimensions de la simulation.

Les tests ont été effectués sur deux machines parallèles :
La machine parallèle du CECPV [2] :

- cluster *Beowulf*
- 30 noeuds bi-processeurs *Itanium 2* cadencés à $1.3GHz$
- $8Go$ de mémoire par noeud
- réseau d'interconnexion *Myrinet 2000*
- environnement MPICH au dessus de Myrinet (débit $\approx 200Mo/s$)

La machine SGI du CINES [3] :

- $SGIOrigin3800$
- 256 processeurs *R14000* cadencés à $500Mhz$
- $384Go$ de mémoire partagée

Le cas test est une simulation d'environ 30 secondes d'un rayon semi-gaussien. Une représentation de cette simulation à différentes itérations est donnée en annexe C.

[1] Yet anOther aDaptive Algorithm
[2] Centre d'Études du Calcul Parallèle et de la Visualisation de l'ULP
[3] Centre Informatique National de l'Enseignement Supérieur

5.1 Accélération

La figure 5.1 montre que notre code a la bonne propriété de conserver une même accélération quelque soit le niveau de détail de la simulation, autrement dit quelque soit la charge de calcul globale et la quantité de données. C'est une propriété intéressante pour l'étude de cas complexes avec des phénomènes très locaux demandant un niveau de détail très fin.

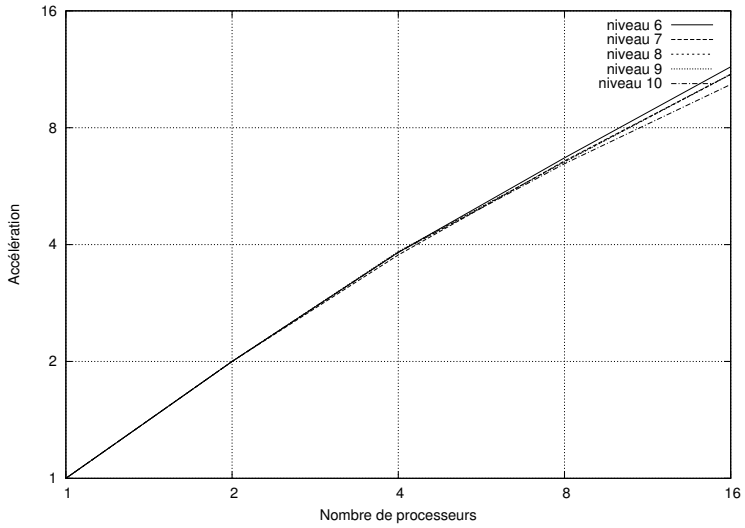


FIG. 5.1 – accélération du code pour plusieurs niveaux de détails

5.2 Efficacité

Sur un seul processeur, notre code est légèrement moins performant que le meilleur code séquentiel existant[4]. La figure 5.2 montre qu'il devient par contre plus rapide dans les simulations avec des détails plus fins. À partir du niveau de détail 10 (donc avec un maximum de 2^{20} cellules), l'accès en temps constant aux cellules par la table de hachage est plus performant que le parcours d'arbre de la version séquentielle.

Donc l'efficacité de notre code sera optimale pour des simulations à fort niveau de détail (de l'ordre de 70% pour 16 processeurs en niveau de détail 10).

[4]YODA

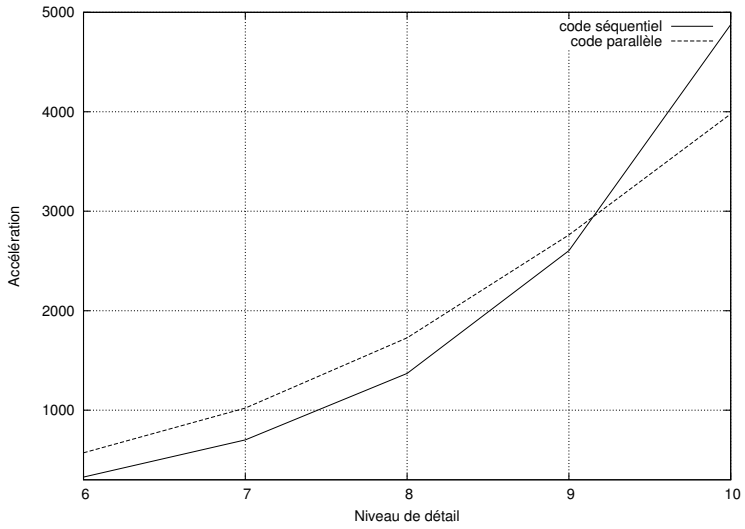


FIG. 5.2 – comparaison code séquentiel – code parallèle

5.3 Extensibilité

L'exécution de notre code sur la machine du CINES nous a permis d'utiliser un nombre plus conséquent de processeur. Nous avons noté (voir FIG 5.3) une baisse significative du temps d'exécution jusqu'à au moins 56 processeurs (le degré de parallélisme maximum n'est donc pas encore atteind).

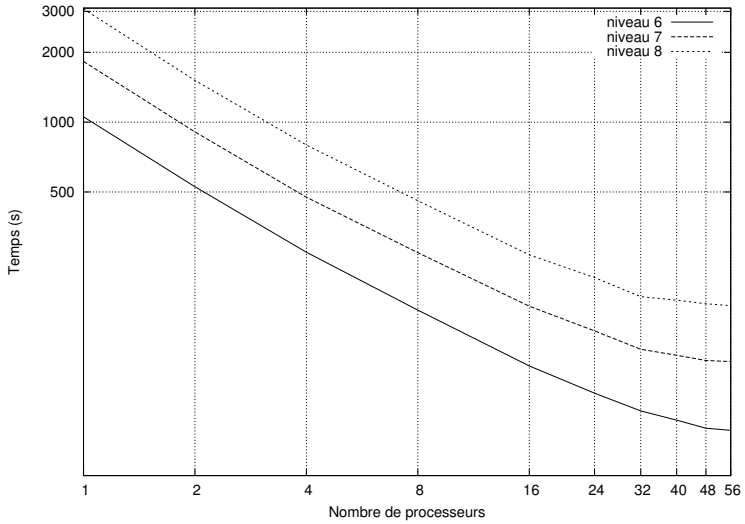


FIG. 5.3 – exécution du code sur un grand nombre de processeur

Conclusion

Dans cette étude, nous avons présenté une parallélisation efficace d'un solveur semi-Lagrangien de l'équation de Vlasov, basé sur un maillage dyadique adaptatif de l'espace des phases. Dans cette optique nous avons proposé un mécanisme d'équilibrage de charge spécifique. Cet algorithme d'équilibrage de charge est dynamique, conserve la localité des données, et permet d'obtenir une bonne accélération, constante malgré l'augmentation de la quantité de données. Dans le cas $2D$, les résultats numériques présentés montrent une efficacité croissante avec le niveau de détails de la simulation. Ce code est donc particulièrement bien adapté aux simulations de grande ampleur, aux détails fins et locaux.

Des optimisations restent encore à implanter, notamment au niveau des communications (l'envoi groupé des messages) et de certaines fonctions utilitaires. La construction des courbes de Hilbert est généralisable à des dimensions supérieurs mais cela reste à implanter. Nous envisageons aussi de fermer cette courbe (courbe de Moore), ce qui devrait donner plus souplesse à la mise à jour des régions. Enfin l'implantation d'un solveur de l'équation de Poisson pour $d > 2$ est le dernier obstacle à l'utilisation de ce code en dimension 4 ou plus.

Les techniques de parallélisation de code et de régulation de charge, développées dans cette étude, prennent tout naturellement place dans le domaine du data-parallélisme. Elles nous ont permis de définir et de conserver la localité des données, ce qui est l'un des enjeux majeur de ce style de programmation.

De futurs travaux pourront porter sur la mise en application de ces méthodes numériques sur des environnements d'exécution hétérogènes, afin de développer des mécanismes prenant en compte les spécificités de la méthode numérique et de la grille de calcul, ce qui n'est actuellement pas ou très peu traité.

Bibliographie

[1] R. Abgrall. Multiresolution representation in unstructured meshes. *SIAM Journal on Numerical Analysis*, 35(6) :2128–2146, 1998.

[2] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, 36(5) :570–580, 1987.

[3] N. Besse, F.Filbet, M. Gutnic, I. Paun, and E. Sonnendrücker. An adaptive numerical method for the vlasov equation based on a multiresolution analysis. In *Numerical Mathematics and Advanced Applications (ENUMATH)*, pages 437–446, 2001.

[4] N. Besse and E. Sonnendrücker. Semi-lagrangian schemes for the vlasov equation on an unstructured mesh of phase space. *J. Comput. Phys.*, 191 :341–376, 2003.

[5] C. K. Birdshall and A.B. Langdon. *Plasmaphysics via computer simulation.* McGraw-Hill, 1985.

[6] M. Campos-Pinto and M. Mehrenberger. Adaptive numerical resolution of the vlasov equation. submitted in Numerical methods for hyperbolic and kinetic problems.

[7] F. Cao, J. R. Gilbert, and S-H. Teng. Partitioning meshes with lines and planes. Technical Report CSL-96-01, Xerox PARC, 1996.

[8] N. Chrisochoides. Multithreaded model for the dynamic load-balancing of parallel adaptive PDE computations. *Applied Numerical Mathematics : Transactions of IMACS*, 20(4) :349–365, 1996.

[9] W. Dahmen, B. Gottschlich-Müller, and S. Müller. Multiresolution schemes for conservation laws. *Numerische Mathematik*, 88(3) :399–443, 2001.

[10] K. D. Devine and J. E. Flaherty. Parallel adaptive *hp*-refinement techniques for conservation laws. *Applied Numerical Mathematics : Transactions of IMACS*, 20(4) :367–386, 1996.

[11] P. Diniz, S. Plimpton, B. Hendrickson, and R. Leland. Parallel algorithms for dynamically partitioning unstructured grids. In *Proceedings of the 7th SIAM Conference of Parallel Processing in Scientific Computing*, pages 615–620, 1995.

[12] Thierry Duboux. *Régulation dynamique du partitionnement de données sur machines parallèles à mémoire distribuée*. PhD thesis, Ecole Normale Supérieure de Lyon, 1996.

[13] F. Filbet. Numerical methods for the vlasov equation. In *Numerical Mathematics and Advanced Applications (ENUMATH)*, 2001.

[14] F. Filbet, E. Sonnendrücker, and P. Bertrand. Conservative numerical schemes for the vlasov equation. *J. Comput. Phys.*, 172 :166–187, 2000.

[15] J. E. Flaherty, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Adaptive local refinement with octree load balancing for the parallel solution of three-dimensional conservation laws. *Journal of Parallel and Distributed Computing*, 47(2) :139–152, 1997.

[16] M. Griebel and G. Zumbusch. Parallel multigrid in an adaptive PDE solver based on hashing. In *Parallel Computing : Fundamentals, Applications and New Directions*, volume 12, pages 589–600, 1998.

[17] M. Gutnic, I. Paun, and E. Sonnendrücker. Vlasov simulations on an adaptive phase-space grid. to appear in Comput. Phys. Comm.

[18] B. Hendrickson and K. Devine. Dynamic load balancing in computational mechanics. *Computational Methods in Applied Mechanics & Engineering*, 184 :485–500, 2000.

[19] O. Hoenen, M. Mehrenberger, and E. Violard. Parallelization of an adaptive Vlasov solver. to appear in PARSIM'04.

[20] Y. F. Hu and R. J. Blake. An optimal dynamic load balancing algorithm. Technical Report DL-P-95-011, Daresbury Laboratory, 1995.

[21] P.K. Jimack. An overview of dynamic load-balancing for parallel adaptive computational mechanics codes. *Parallel and Distributed Processing for Computational Mechanics : Systems and Tools*, pages 350–369, 1999.

[22] M. T. Jones and P. E. Plassman. Computational results for parallel unstructured mesh computations. Technical Report UT-CS-94-248, Argonne National Laboratory, 1994.

[23] J. K. Lawder and J. H. King. Using space-filling curves for multi-dimensional indexing. *Lecture Notes in Computer Science*, 1832 :20–35, 2000.

[24] Official mpi (*Message Passing Interface*) standards documents, errata, and archives. http ://www.mpi-forum.org/.

[25] M. Parashar, J. C. Browne, C. Edwards, and K. Klimkowski. A common data management infrastructure for adaptive algorithms for PDE solutions. In *Supercomputing '97*, 1997.

[26] A.K. Patra and J.T. Oden. Problem decomposition strategies for adaptive hp finite element methods. *Computing Systems in Engineering*, 6(2) :97–109, 1995.

[27] J. Pilkington and S. Baden. Partitioning with spacefilling curves. Technical Report CS94-349, Dept. Computer Science and Engineering, University of California, 1994.

[28] E. Pramono, H. Simon, and A. Sohn. Dynamic load balancing for finite element calculations on parallel computers. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 559–604, 1995.

[29] Martha Rosa Castaneda Retiz. *Étude quantitative des mécanismes d'équilibrage de charge dans les systèmes de programmation pour le calcul parallèle.* PhD thesis, Institut National Polytechnique de Grenoble, 1999.

[30] H. D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2 :135–148, 1991.

[31] E. Sonnendrücker, F.Filbet, A. Friedman, E. Oudet, and J.L. Vay. Vlasov simulation of beams on a moving phase-space grid. to appear in Comput. Phys. Comm.

[32] E. Sonnendrücker, J. Roche, P. Bertand, and A. Ghizzo. The semi-lagrangian method for the numerical resolution of vlasov equations. *J. Comput. Phys.*, 149 :201–220, 1999.

[33] E. Violard. A semantic framework to address data locality in data parallel languages. *Parallel Computing*, 30 :139–161, 2004.

[34] E. Violard and F. Filbet. Parallelization of a vlasov solver by communication overlapping. In *PDPTA'02*, 2002.

[35] C. Walshaw and M. Berzins. Dynamic load-balancing for PDE solvers on adaptive unstructured meshes. *Concurrency : Practice & Experience*, 7(1) :17–28, 1995.

[36] C. Walshaw, M. Cross, S. Johnson, and M. Everett. JOSTLE : Partitioning of unstructured meshes for massively parallel machines. In *Parallel Computational Fluid Dynamics : New Algorithms and Applications*, pages 273–280, 1995.

[37] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Supercomputing*, pages 12–21, 1993.

Annexe A

Algorithme parallèle détaillé

▷ Prédiction du nouveau maillage temporaire $\widetilde{\mathcal{M}}^{n+1}$ (initialement vide)

POUT TOUTE cellule $\alpha \in \mathcal{M}^n$, locale à un processeur p **FAIRE**

- déterminer son centre c_α
- calculer le point $\bar{c_\alpha} = F_n^{wd}(c_\alpha)$ résultat de l'advection en avant de c_α
- trouver la région à laquelle appartient $\bar{c_\alpha}$ et l'identifiant du processeur p' correspondant
- créer l'unique cellule $\widetilde{\alpha}$ de niveau $|\alpha|$ qui contient $\bar{c_\alpha}$ dans $\widetilde{\mathcal{M}}^{n+1}$

SI $p' = p$ **ALORS**

- ajouter $\widetilde{\alpha}$ à $\widetilde{\mathcal{M}}^{n+1}$ et les autres cellules nécessaires pour garantir la consistance du maillage
- raffiner $\widetilde{\alpha}$ d'un niveau (si $|\widetilde{\alpha}| \neq J$)

SINON

- envoyer $\widetilde{\alpha}$ au processeur p

FIN SI

FIN POUR TOUT

POUR TOUT processeur $p' \neq p$ **FAIRE**

- envoyer le message *end-of-send* à p'

FIN POUR TOUT

TANT QUE on a pas reçu $P - 1$ messages *end-of-send* **FAIRE**

- réceptionner une cellule α en provenance de p'
- ajouter α dans $\widetilde{\mathcal{M}}^{n+1}$ et les autres cellules pour assurer la consistance du maillage
- raffiner α d'un niveau (si $|\alpha| \neq J$)

FIN TANT QUE

▷ Évaluation de la fonction de distribution

POUT TOUTE cellule $\alpha \in \mathcal{M}^{n+1}$, locale au processeur p **FAIRE**
  POUR TOUT noeud $a \in \alpha$ **FAIRE**
    • calculer le point résultat de l'advection arrière de a : $\bar{a} = B_n^{wd}(a)$
    • déterminer le processeur p' qui contient $\bar{a}$
    SI $p' = p$ **ALORS**
      • interpoler la valeur de $\bar{a}$ (i.e calculer $f^n(\bar{a})$)
      • ajouter le noeud a et sa valeur $f^n(\bar{a})$ dans le maillage $\mathcal{M}^{n+1}$
    SINON
      • envoyer le point $\bar{a}$ au processeur p'
      • ajouter le noeud a dans le maillage
      • initier la réception en provenance de p' de la valeur de $\bar{a}$
    FIN SI
  FIN POUR TOUT
FIN POUR TOUT

POUR TOUT $p' \in P, p' \neq p$ **FAIRE**
  • envoyer le message *end-of-send* à p'
FIN POUR TOUT

TANT QUE on a pas reçu $P - 1$ messages *end-of-send* **FAIRE**
  • receptionner un point $\bar{a}$ en provenance de p'
  • calculer la valeur de $\bar{a} = f^n(\bar{a})$
  • renvoyer cette valeur à p'
FIN TANT QUE

On a à présent un premier $\widetilde{\mathcal{F}}_{n+1}$.

$\triangleright$ Compression : $\widetilde{\mathcal{F}}_{n+1} \to \mathcal{F}_{n+1}$

POUR TOUT groupe de 4 cellules $\in \mathcal{M}^{n+1}$ issues de la même cellule mère de niveau inférieur, local au processeur p **FAIRE**

- reconstruire α la mère de ces cellules
- calculer : $d(b) := \widetilde{f}^{n+1}(b) - I(b, \alpha, \widetilde{f}^{n+1}(\alpha))$ pour $b \in \Gamma^2(\mathcal{D}(\alpha)) \backslash \Gamma^2(\alpha)$
- remplacer le groupe de cellules dans $\widetilde{\mathcal{M}}^{n+1}$ par α si le test de compression (2.2) est vrai

FIN POUR TOUT

Les valeurs restantes sont celles de $\mathcal{F}_{n+1}$

Pour passer à l'étape de temps suivante il suffit maintenant de remplacer $\mathcal{M}^n$ par $\mathcal{M}^{n+1}$ et de recalculer les valeurs du champ E.

Annexe B

Construction d'une courbe de Hilbert multi-résolution

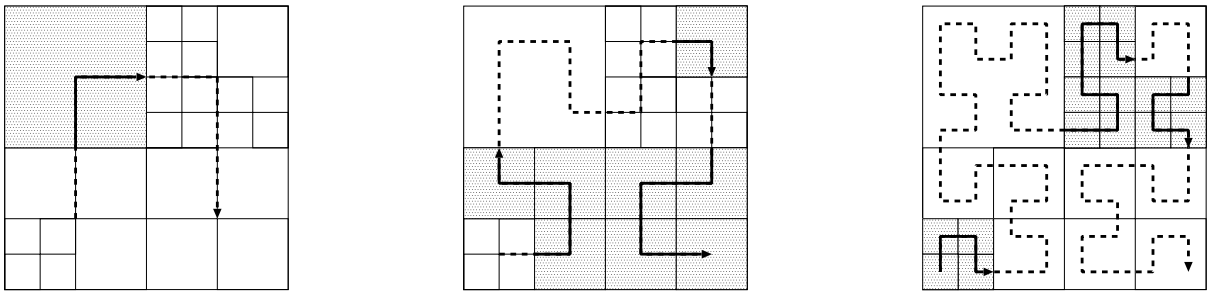


FIG. B.1 – construction de la courbe de Hilbert par niveaux successifs

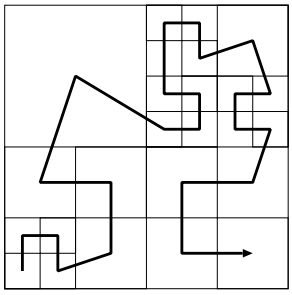


FIG. B.2 – courbe de Hilbert multi-résolution

Annexe C

Rendu d'une simulation

Chaque figure représente l'état de la simulation à une étape de temps donnée :

1. l'image du haut représente les valeurs de la fonction de distribution, pour le cas d'un rayon semi-gaussien.
2. l'image de gauche représente le maillage adaptatif, les mailles étant colorées selon les valeurs de la fonction de distribution.
3. l'image de gauche représente l'affectation des différentes régions aux 8 processeurs utilisés pour cette simulation.

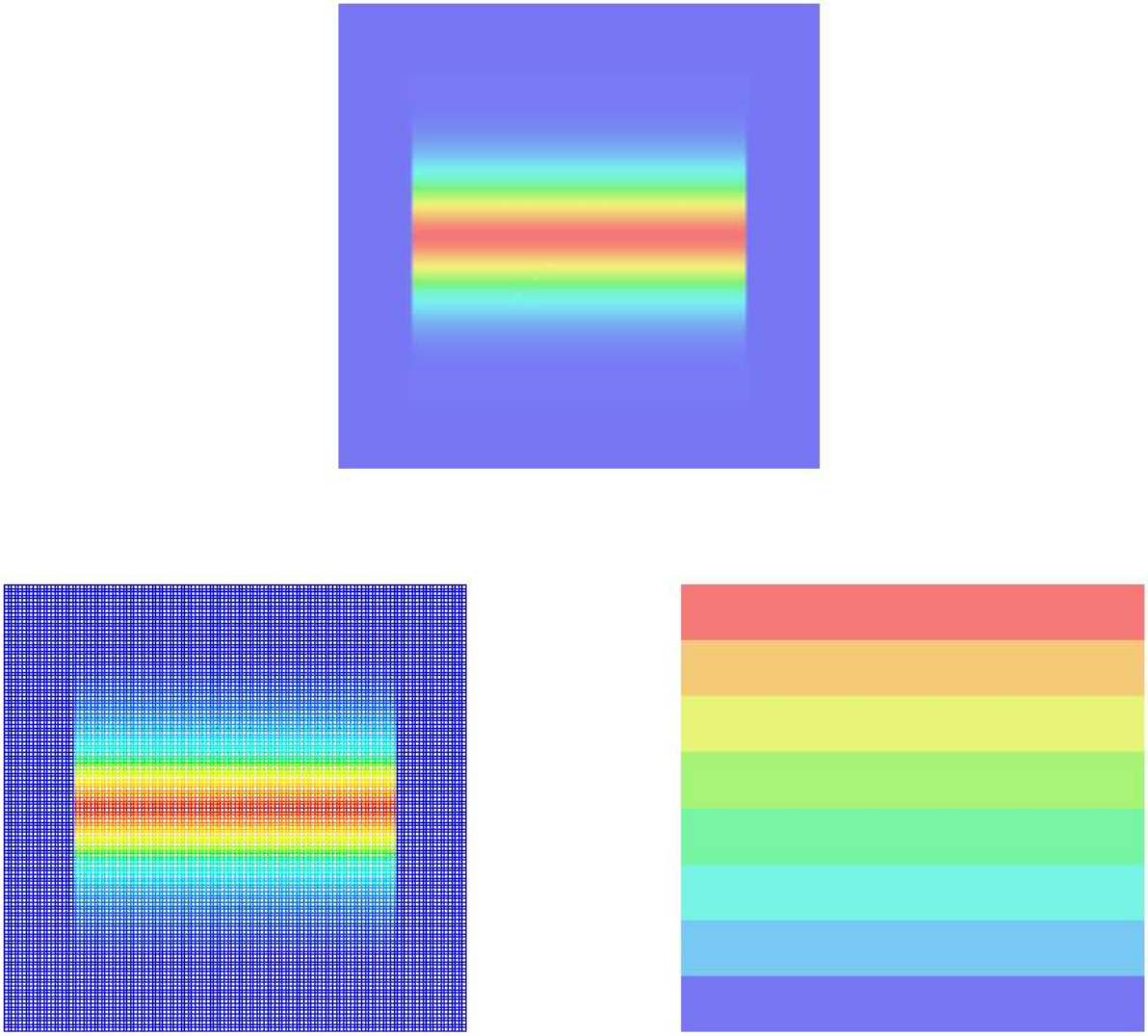


FIG. C.1 – Initialisation

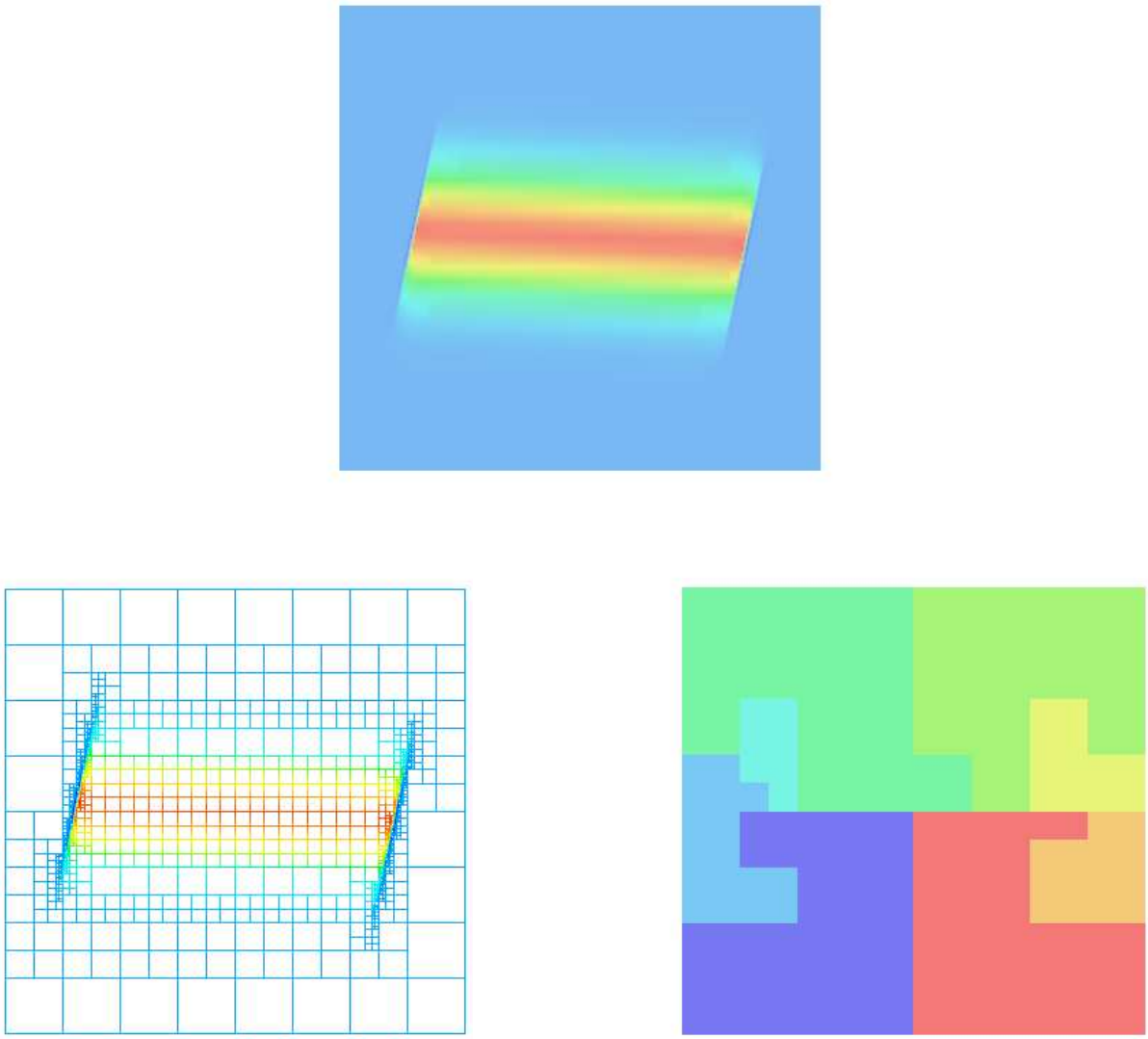


FIG. C.2 – Création des régions à la 2e itération

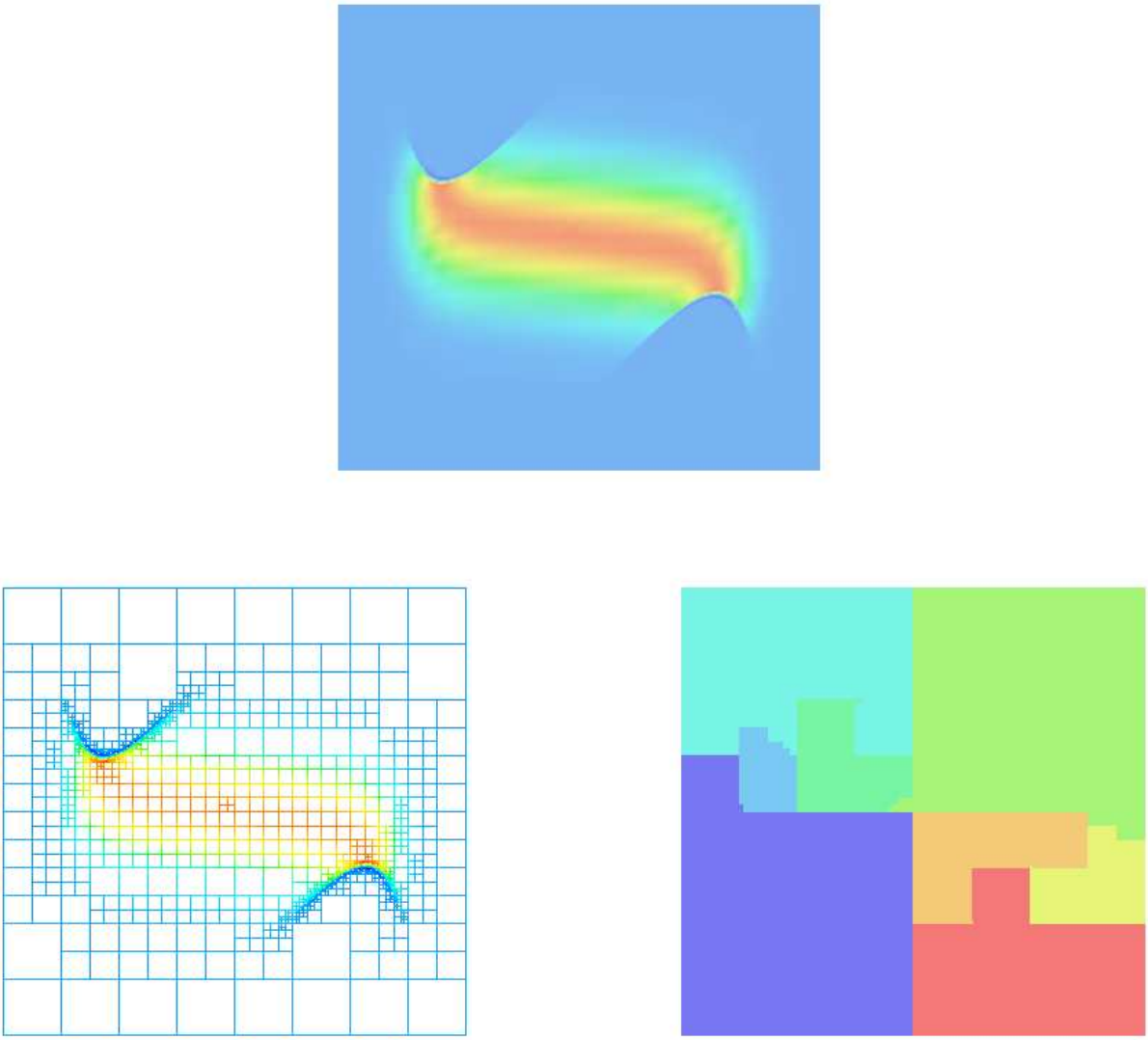


FIG. C.3 – Mise à jour des régions (10^{e} itération)

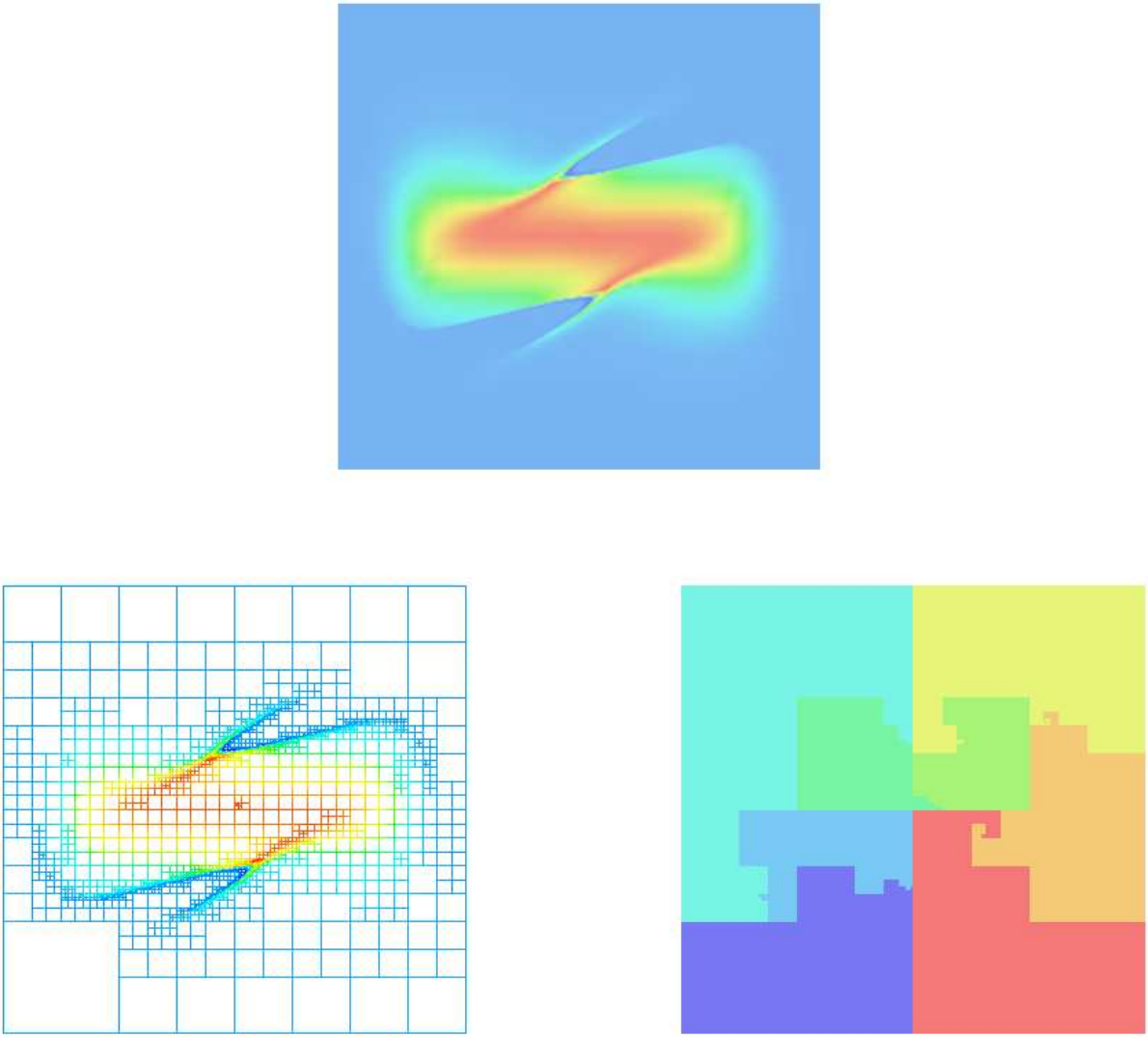
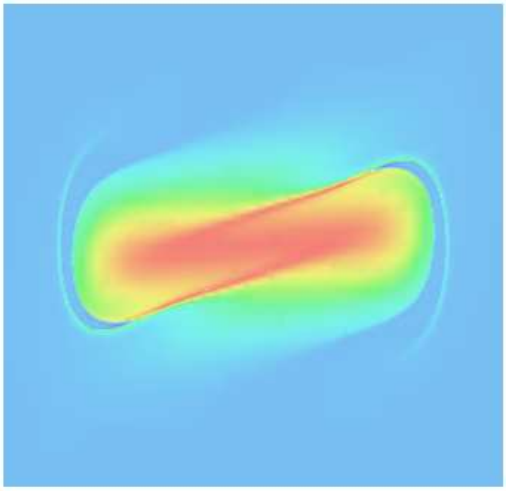
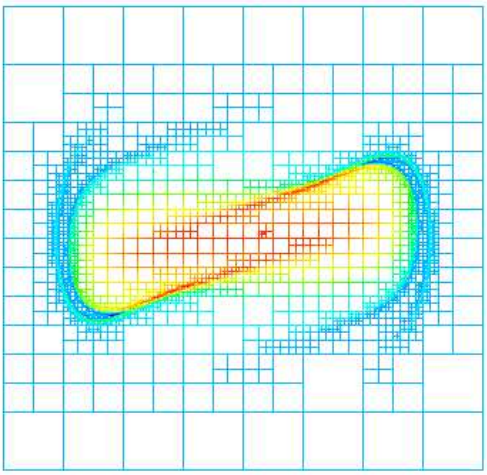


FIG. C.4 – 30e itération





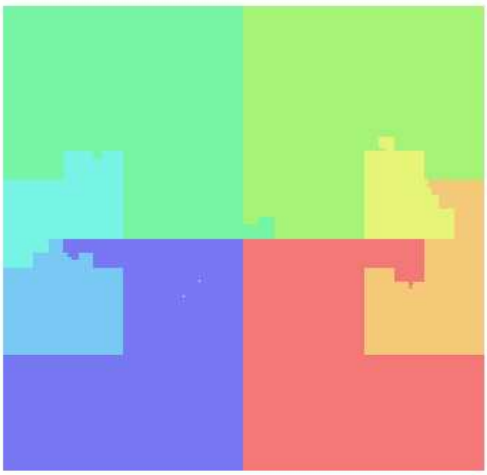


FIG. C.5 – 50e itération